第一部分

图形编程

第一章 Windows

Windows 是一个实时的操作环境，一旦我们理解了这个特殊环境的基础，就能充分利用 windows 所拥有的各种特点，来编制 Visual Basic 的动画应用程序。

Windows 运行程序时，程序也在运行 Windows。如图 1.1 所示。一种基于消息的机制使应用程序与 Windows 建立相互联系。

1.1 Windows 如何运行程序

Windows 在一场有许多参与者的游戏中充当仲裁人，而用户的应用程序就是这些参与者之一。Windows 通过以下三种服务给用户程序提供良好的运行环境。第一：Windows 装入并执行用户程序。这叫启动用户程序。第二：Windows 对计算机的内存进行管理，这样，用户程序（同时任何别的 Windows 应用程序也能运行）能够访问所需的内存或别的系统资源。第三：Windows 管理大部分用户程序所需的低级输入。

1.1.1 启动应用程序

当用户选定了应用程序的图标后，Windows 为用户程序分配一部分内存。然后装入执行代码。并建立一个本地栈。程序在运行时再装入所需的数据或别的特殊资源。这些资源是 Windows 环境独有的。它们包括系统菜单加速键、对话框，信息框、图标或位图等。不要将 Windows 资源与系统资源相混淆，后者包括内存、磁盘存取、键盘或鼠标输入、视频存取。

1.1.2 管理系统资源

如果多个 Windows 应用程序同时运行，内存便可能要受到限制。为此 Windows 提供一个附加的内存管理器，它在后台操作，能使每个应用程序得到其所需的内存。这意味着有时得将内存中的代码、数据和资源从内存中送出去以便给另一个应用程序让出空间。代码、数据和资源中每一种都按单元来组织，这些单元叫段。Windows 管理内存时对一整个段进行交换或释放。

1.1.3 管理输入

为了使应用程序具有硬件无关性，Windows 提供一套完整的输入和输出服务。这使用户开发应用程序时比较容易，从而没有必要为不同的计算机显示器，打印机等加上特殊的说明。如图 1.1 所示，Windows 不停地查询键盘，鼠标或定时器，当上述设备发生一个事件时，Windows 产生一个相应的消息并将它放入系统队列中。系统队列是一先进先出的消息队列。这些消息与从键盘和鼠标接收的事件相对应，定时器事件单独存放。

Windows 能够识别哪些消息用于哪一应用程序，甚至多个应用程序同时运行时也是如此。别的应用程序（或 Windows 本身）在运行时能直接向用户程序发送消息。送给用户程序

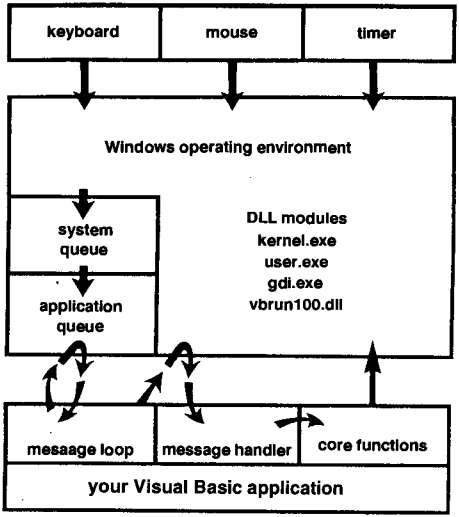

图 1.1 运行时刻用户应用程序与 Window 环境进行交互的概念图示

的消息存放在 Windows 为此用户程序专门保留的一个独立队列中。

消息参数

理解消息是理解 Windows 编程的关键。用户程序通过从 Windows 专门为它保留的队列中获取消息而获得输入。Visual Basic 能自动地在用户程序中生成用来获取或分发消息的代码。用户程序可以向别的应用程序或 Windows 本身发送消息，也能从别的应用程序或

Windows 本身接收消息。再说一遍，Visual Basic 在这方面已经为你做好了一切。

每个消息都是一个统一的数据集合。存于 Windows 的内部数据结构 MSG 中。MSG 包括 6 个成员：hwnd、WM、wParam、lParam、time 和 pt。

hwnd **窗口句柄**

MSG 的第一个成员 hwnd 用来标明需接收消息的用户程序。每个 Windows 的应用程序都必须有一个主窗口（或者称为显示窗口）。Visual Basic 编程者将主窗口称作主框架，通常用显示屏上一矩形区域来实现。每个窗口（或叫框架）通过一个叫句柄的令牌来标识，写成 hwnd。MSG 的第一个成员包含一个 hwnd 的窗口句柄。

WM_**消息标识符**

第二个成员用于标识所发生事件的类型，叫消息标识符。每个消息标识符冠以前缀 WM_windows message 的缩写。如用户从用户程序主菜单中选择了某个命令，Window 将发送一个 WM_COMMAND 消息给程序。

wParam **和** lParam

MSG 的第三个成员叫 wParam，第四个成员叫 lParam。MSG 的第三个和第四个成员包括与消息有关的别的参数。在 WM_COMMAND 消息的例子下，第三个成员指出哪一个键被按下。当接收到一个 WM_MOUSE 消息时，表示用户移动了鼠标。这种情况下，MSG 的第四个成员包括了鼠标光标的新坐标值。

time **和** pt

MSG 的第五和第六个成员用得较少。第五个成员叫 time，表示消息放入队列中的时间。第六个成员叫 pt，包括了 Windows 将消息放入队列时鼠标光标的坐标值。

1.1.4 *消息循环*

用户程序使用从消息队列取来的消息。为了读取消息，Visual Basic 在用户程序中安排了一个特别的循环，称为消息循环，消息循环由用户程序框架模块进行管理。

如果用户程序检测到了消息队列中的一个消息，便从消息循环中跳出来，执行已经编制好的动作。然后程序控制再转回消息队列并开始检测下一个到来的消息。

用户程序的消息循环调用 Windows 现成的函数 GetMessage()，它用来指示 Windows 进行消息检查。当它被调用时，Windows 首先在应用程序队列中搜寻消息，然后在系统队列中搜寻键盘和鼠标消息，再检查当前是否有定时器消息。

1.1.5 *消息句柄*

当用户程序要检测一个消息的到来和调用编入程序中的 Visual Basic 过程（函数和子过程）时，是通过 Windows 向用户程序的一个叫消息句柄的专门部分发送消息完成的。每个 windows 应用程序都必须有一个消息句柄，这是因为 Windows 通过消息句柄与每一个窗口相联系，消息句柄也叫窗口过程。Visual Basic 将在用户程序中自动插入完成此功能的相似代码。

Windows 不允许用户程序直接调用别的函数，这是因为 Windows 是一协同多任务环境，可以同时运行多个应用程序。当别的应用程序要向当前的用户程序发送消息时，最有效的办法便是向用户程序的消息句柄发送消息。用户程序马上就可作出反应。如果将消息送

入 Windows 为用户程序所保留的队列时，此消息可能排在一长串别的消息的后面，这样用户程序要经过等待才能取到此消息，而当消息直接送入消息句柄时，其间则没有等待。用户程序要要发送消息时，可调用 Windows 现成的函数 Send Message()和 Postmessage()。

1.2 用户程序如何运行 Windows

用户程序执行的许多操作都是通过调用 Windows 销售版中现成的函数来实现的。在大多数情况下，Visual Basic 在源程序代码和 Windows 中可被调用的函数之间充当中介。在某些情况下，用户可能宁原直接调用 Windows 的函数，但这样的话，用户就用不上 Visual Basic 所提供的扩展参数检查的特性了。

这些可被调用的 Windows 例程都在 Windows 的应用程序接口(API)中，当用户程序调用这些函数的时候，实际上就是在运行 Windows。当用户使用 Visual Basic 所提供的不同函数、说明或特性时，是 Visual Basic 在运行 Windows；如果直接调用 Windows API，则是用户在运行 Windows。

用户程序可用 API 来完成许多不同的任务。用户可直接或通过 Visual Basic 来使用 API。用户程序可在显示屏上放大、缩小或移动窗口；可以重新设置菜单系统；可将一幅图像拷入 Windows 的剪裁板中，然后将该图像贴入正在运行的别的应用程序中；可以操作 Windows 的打印机管理中，使它在用户打印机上进行图形的硬拷贝；也可使用 API 在用户程序的主窗口中产生图形，包括动画。

1.2.1 动态链接库

Windows API 中的函数包含在叫动态链接库(DLL)的运行库中。Windows 销售版中提供三个 DLL。分别为 kernel. exe user. exe 和 gdi. exe 如图 1.1 所示。

Visual Basic 本身也当作一个 DLL 得以实现。用户用 Visual Basic 编制的可执行程序在运行时需要调用一套特别的例程。这些例程包含在叫 vbrun 100. dll 的动态链接库中。这个 DLL 的作用是充当用户的 Visual Basic 程序与 windows 的三个 DLL 的接口，只有合法的 Visual Basic 用户才有权销售与其所编的 Visual Basic 应用程序一起的 vbrun. dll 的拷贝。

kerner. exe DLL 提供一般的窗口管理函数，以便支持窗口系统。user. exe DLL 提供象内存管理、多请求管理的系统服务。图形设备接口，即 gdi. exe 提供一套图形例程，其中很多例程非常适合于制作动画。本书中的 Visual Basic 例程利用了 gdi. exe 图形生成器中的许多可调函数。

1.3 建立 Visual Basic 应用程序

开发和测试 Visual Basic 应用程序差不多就是使用装配模块。开发者先通过 Visual Basic 的图形和交互接口选择这些装配模块，然后将它们装配成一个应用程序原型，然后再将源代码连接到程序原型中，以便完成所设计的任务。

1.3.1 应用程序装配模块

Visual Basic 提供 6 个基本的装配模块：

- 框架
- 控制
- 特性
- 事件
- 方法
- 菜单

框架(forms)

框架就是窗口，用户使用 Visual Basic 开发的每个应用程序都必须有一个主框架，这叫主窗口或显示窗口。Visual Basic 将自动地为应用程序产生主框架。在 Visual Basic 的编程中，一个框架(窗口)就是一个对象。

控制

控制就是主框架中用来接收用户输入和显示输出的图形对象。Visual Basic 中控制的例子有命令按钮，列表框、文字框和图形框。每个控制本身还包含可识别的属性、事件和方法。象框架一样，在 Visual Basic 的编程中，控制也是一个对象。

属性

属性是指一个框架或一个控制的属性。它用于定义一个框架或控制的大小、颜色和位置，也可用于描述一个框架或控制的行为，比如是否可视或是否使能。如以下语句：

This Picture. Visible = True

表明名为 ThisPicture 的图形框为可视。

事件

一个事件是能被框架或控制认可的行动，比如点动鼠标或击键。用户可以自己编写代码块，当框架或控制检测到事件时便自动调用它，这部分代码块便叫事件过程。

例如，当用户在菜单中名为 IDM_MenuItem 上点动时程序将自动调用。一个叫 IDM_MenuItem_Click()的事件过程。事件过程通常都是子过程，且无返回值。Visucal Basic 所支持的别的过程是函数过程，有返回值。

方法

方法与一个标准的 Visual Basic 函数或语句类似，只不过它对对象进行操作。Visual Basic 的对象是框架或控制。方法也可用于 Visual Basic 支持的一些专门对象，如打印机和剪裁板。例如，以下说明：

Form1. Hide

表示将主框架从屏幕上移去的消隐方法。和上面在属性中讨论的将 Visible 置成 False 有同样效果。

菜单

用 Visual Basic 的菜单设计窗口能形成菜单及菜单条。单独的菜单条叫菜单控制。当用菜单设计窗口来设计应用程序的菜单系统时，用户需为每一菜单条取一个控制名字，Visual Basic 通过此名字来调用用户所提供的事件过程。例如，用户为一菜单条取名 IDM_

ThisItem，则用户每次点动此菜单条时，Visual Basic 都将调用名为 IDM_ThisItem_Click() 的事件过程，作为编程者，则要负责提供名为 IDM_ThisItem_Click() 的事件过程的代码。Visual Basic 提供含有事件过程首尾说明的代码模板，只有用户在 Visual Basic 提供的 Sub 和 End Sub 之间加入自编的各语句，此事件过程才会有具体的响应。

1.3.2 使用 Visual Basic

使用 Visual Basic 就是使用它所提供的四种窗口：

- 工程窗口
- 框架定时设计窗
- 菜单设计窗
- 代码窗

使用工程窗

工程窗包括组成用户程序的所有框架和模块的列表。用于 Visual Basic 某应用程序的工程窗如图 1.2 所示。

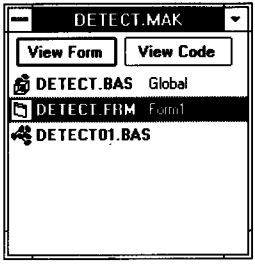

图 1.2 Visual Basic 工程窗口。本书中各个示例程序由一系列全局格式及功能模块构成

使用框架定时设计架窗

框架定时设计窗是用户程序显示窗的图形实现，当设计应用程序时，需将控制如定时器和图形框置于框架定时设计窗中。框架定时设计窗如图 1.3 所示。其中的栅格便于用户在此窗中放置控制块。

用户使用菜单设计窗为框架定时设计窗中的应用程序设计原型菜单系统。Visual Basic 编程时框架定时设计窗中的原型菜单如图 1.4 所示。

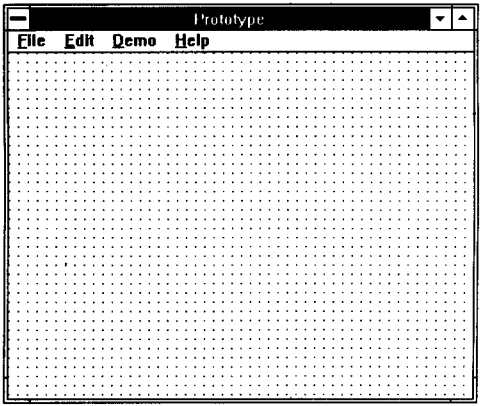

图 1.3 Visual Basic 格式设计窗口

添加用户代码

框架代码是指框架中事件过程或声明的代码，Visual Basic 将它以扩展名.frm 存于定义它的目录中。当用户编写某一事件发生所要调用的代码时，就将代码加于某个特定框架或控制块中。图 1.5 所示即为代码窗。当用户选中应用程序菜单条 IDM_Clear 时，名为 IDM_Clear_Click()的子过程将被调用。

函数代码由用户提供的用于完成某些任务的子过程或函数过程组成，函数代码一般存于独立于.frm 模块的.bas 模块中，.frm 模块中的过程能调用.bas 模块中的过程，反之则不行。

用 Visual Basic 进行原型设计

当用户编制，测试和再测试应用程序时，就是在进行原型设计工作，Visual Basic 使用户进行此项工作变得非常容易，因为可以运行应用程序然后快速返回 Visual Basic 环境。图 1.6 所示为 Visual Basic 交互编程环境的 Run 菜单，选用 Run 将在 Visual Basic 环境的控制之下启动应用程序。选 Set Startup Form 将显示图 1.7 所示的对话框，用来标识用户程序运行

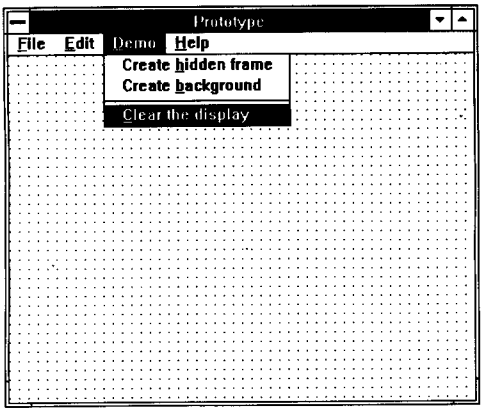

图 1.4 本书中的应用程序菜单系统统一,以便用户将精力集中于所讨论的图形上

的起点。

使用菜单设计窗

通过图 1.8 所示的 Window 菜单可以激活图 1.9 中所示的菜单设计窗,因此用户可以给应用程序中的菜单条命名,其中也包括每个下拉菜单的菜单条。例如,在图 1.9 中,Caption 中包含将出现的菜单条的名字,CTLName 中就是当事件发生时 Visual Basic 将调用的控制块名字(见图 1.5)。

用户还可通过菜单设计窗给菜单系统添加热键,当程序执行时,用户可按下热键从而激活菜单中相应的某命令。例如,本书中用热键 F1 来激活 Help 消息框,与选中命令单中 Help 条有同样效果。Windows 编程者也将它称作加速键。

菜单设计窗还支持键存取,当用户在菜单名中插入与操作符,例如 Th&Menu 时,可以按下 Alt 键并按 S 键来激活此菜单。如果与操作符插在菜单条的名字中,例如 IDM_Menu&Item,用户只需要按 I 键即可激活此 IDM_MeniItem 命令。因为本书中的例程中采用

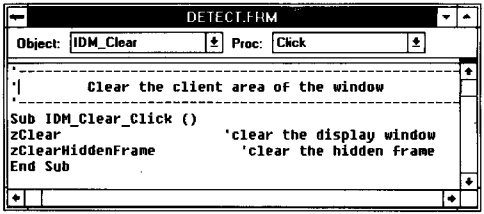

图 1.5 一个 Visual Basic 代码窗口，此处所示的是用户使用鼠标单击名为 IPM_clear 菜单条时的调用代码

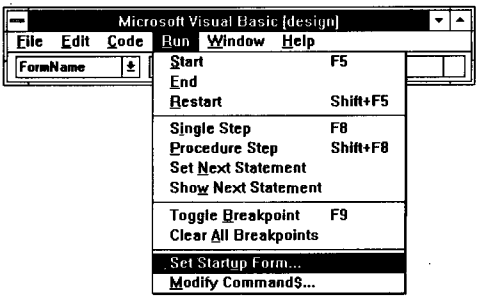

图 1.6 Visual Basic 的 Run 菜单。书中的每个示例程序使用了一个固定的启动模块，该模块在选择 Set Startup Form 菜单条时被激活

了键存取，故用户可用鼠标或键盘来运行程序，Windows 编程者也将它称作助记键。

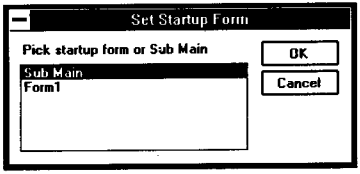

图 1.7 Visual Basic 的 Set Startup Form 对话框，它允许用户指明在运行用户应用程序时首先调用哪一个模块

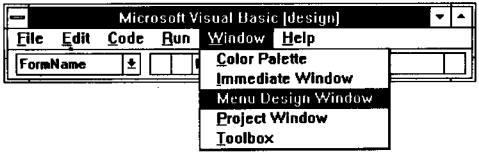

图 1.8 Visual Basic 的 Window 菜单。Menu Design Window 菜单条允许用户构造自己的应用程序的菜单系统

1.3.3 用 Visual Basic 进行多模块编程

开发 Visual Basic 应用程序就是进行多模块编程，在标准的 Visual Basic 应用程序中使用了种类型的模块。

- 全局模块
- 框架模块
- 功能模块

使用全局模块

应用程序的全局模块包含出现在所有模块中的常量说明、变量说明和过程说明。每个 Visual Basic 应用程序都有一个全局模块。本书例程中的全局模块用于说明代码部分将要调用的 Windows API 中的过程。

图 1.10 所示为用于全局模块的代码窗，全局模块包含整个程序出现的常量和变量，其中包括使用户能直接调用 Windows API 函数的说明。对 GDI 的访问使显示程序具备了高级动画功能的基础。全局模块使用扩展名.bas。

使用框架模块

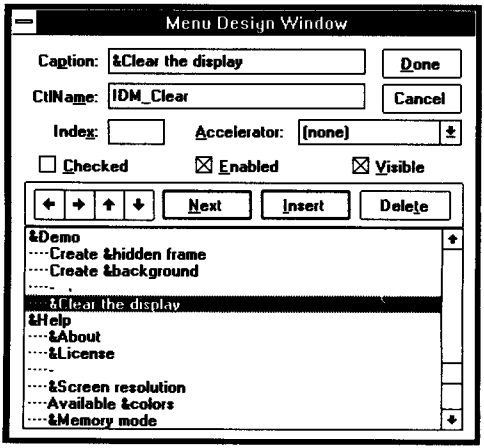

图 1.9 Visual Basic 提供的 Menu Design Window 选择。书中的每个示例程序具有同一的菜单系统

图 1.11 所示为用于框架模块的代码窗，框架模块包括某事件发生时(如点动鼠标)将自动调用的事件过程的源代码。它还包括应用程序主框架和菜单系统的属性。框架模块以一种 Visual Basic 能读的特殊格式存贮，使用扩展名.frm。当用户希望打印一个框架模块文件时，需先用 Visual Basic 将它转换成.txt 文件。

使用功能模块

图 1.12 所示为用于功能模块的代码窗。功能模块包括在框架模块中的事件过程要完成专门任务时所需调用的例程(函数和子过程)。注意功能模块中的过程不能调用框架模块中的过程。Visual Basic 应用程序可以设置成多个功能模块。本书中大部分例程包含两个功能模块。第一个功能模块包含启动代码，用于初始化全局变量(参见图 1.6，图 1.7)。第二个模块包含一般的过程。功能模块用文件扩展名.bas，象框架模块一样，当要打印它时，需先将它转成.txt 文件。

当这三种模块：全局、框架、功能模块与 Visual Basic 所具有的高级编程特点相结合用于编制 Windows 的图形应用程序时，将给用户带来想象不到的通用性，这一点，在后续各章的

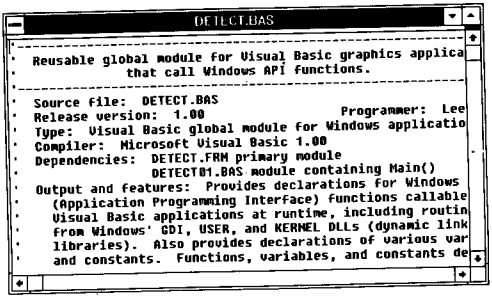

图 1.10 一个 Visual Basic 代码窗口。这里所示的是一个包含应用程序中全体可视常量和变量的典型的全局模块，包括调用 GDI 图形函数的声明

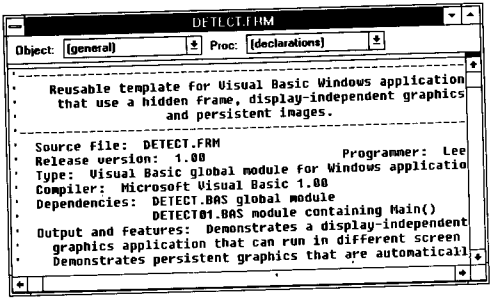

图 1.11 一个 Visual Basic 代码窗口。这里所示的是一个典型的格式模块，包含调用如鼠标单击事件反应的过程代码

例程中将看到。

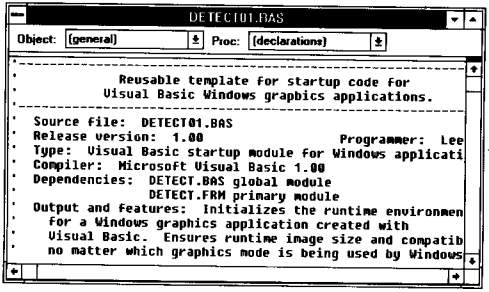

图 1.12 一个 Visual Basic 代码窗口，这里所示的是一个典型的功能模块应用程序在开始执行时调用启动模块

Microsoft Windows 提供一个现成的图形库叫图形设备接口(GDI)。用户的 Visual Basic 应用程序运行时能调用 GDI 的函数。这些可调的函数都包含在 Windows 销售版的动态连接库之一 gdi. exe 中。

GDI 提供强有力的图形例程,而且为了适用于 Windows,还经过了优化,除了标准的直线,抛物线和填充例程外,GDI 还提供一套可供选择的混色,位块传送和剪裁函数,这三类函数特别适合于高级的图形工作。位块传送函数在用 Visual Basic 应用程序来制作动画序列时更是特别有用。

2.1 GDI

GDI 是以 Windows 为宿主的图形发生器,任何在 Windows 环境下运行的应用程序都能调用 GDI 函数。GDI 能直接将图形输出至屏幕、内存、打印机、磁盘文件和别的设备。正是因为输出的不定性,故用户每次调用 GDI 中某函数时需提供一个场境。场境就是用户与 GDI 商定输出设备的一套假定。设备这个词对 GDI 来说有特定的含义。

2.1.1 设备

设备是输入和输出的焦点。键盘、鼠标、定时器是输入设备,显示屏、打印机、晒图机、磁盘驱动器和调制解调器是输出设备。输出设备也包括内存中的仿真设备,如位图和元文件。位图是内存中一仿真的绘画平面。元文件是 GDI 调用函数的集合,它们能够用于存贮重建某图像的方法。

在调用 GDI 函数前,用户必须向 GDI 提供设备场境信息,即告诉 GDI 怎样将图像输出及输往何处。用户通过创建一设备场境来实现这一点。

2.1.2 设备场境

一个设备场境代表一个输出设备及其设备驱动程序。设备驱动程序就是直接控制硬件的软件。打印机有设备驱动程序、显示器、鼠标也都有。

如果用户在一台配有 VGA 的计算机上运行 Windows,用来控制显示内存和图形适配器的软件位于一个名叫 VGA. drv 的驱动程序中,通常此驱动程序在硬盘的子目录 C:\windows\system 个。起动时,Windows 将此设备驱动程序装入内存,当用户程序调用 GDI 在屏幕上产生一幅图像时,GDI 调用 VGA. drv 中的例程来控制 VGA 图形适配器,这里的 VGA 图形适配器就是设备,而 VGA. drv 就是驱动程序。

GDI 通过所谓的句柄来识别设备场境。Visual Basic 编程语言支持 hDC 属性,它用于格式,图形框和打印机。HDC 属性就是一个句柄。

设备场境可以描述很多不同的设备,显示场境就是设备场境的一种。

2.1.3 显示场境

显示场境用于详细描述显示屏上一个窗口，特别是描述怎样将图形输出写到某特定窗的用户区中。用户区是窗口的中间部分，或绘图面(不包括标题框，菜单条和边界)。Windows 编程者常用缩写 DC 指示显示场境或者设备场境。

2.2 显示场境

当用户使用 GDI 为某一窗口初始化其显示场境时，Windows 将图形原点复位至窗口用户区的左上角，将剪裁复位为整个用户区。图形原点就是坐标(0,0)所描述的象素位置。剪裁区就是用来画图的视口。Windows 将在视口的边缘剪裁图像，所绘图像不会落在剪裁区的外面，图形原点和剪裁区仅仅是显示场境许多缺省属性中的两个。

2.2.1 缺省属性

图 2.1 中简要地列出了缺省的 DC 属性，尽管这些属性中每个都可以被用户的 Visual Basic 应用程序修改，许多 DC 都用它们的缺省值。

缺省的 DC 属性	
背景颜色	白色
背景模式	不透明
画笔颜色	白色
笔颜色	黑色
正文颜色	黑色
字体	系统字体，系统比例
剪裁区域	定义区
DC 圆点	0,0(左上角，定义区)
笔位置	0,0
画笔位置	0,0

图 2.1 除非另外指定，否则用户图形应用程序的显示窗口显示窗口将使用这些缺省属性

- 缺省的背景色为白色，即 GDI 假定窗口的用户区背景色为白色。
- 缺省画刷颜色为白色，所有 GDI 所现的实体如矩形，将填充为白色。
- 缺省画笔颜色为黑色，GDI 画的所有直线，矩形，椭圆，多边形在白背景下呈现黑色。
- 缺省字色为黑色，GDI 在白色背景上写黑色字符。
- 背景模式为不透明的。每个字符图元所占的背景区为不透明的，将覆盖或擦掉屏幕上已有的图形。画黑色的虚线或点划线时，点划之间的空白将覆盖任何已有的图像。
- 缺省的剪裁区为整个用户区。当在剪裁区以外画物体时不可见，所有的直线、矩形、位块和文本都将在用户的边缘进行剪裁

· 缺省的图形坐标原点为用户区的左上角，且当前画笔或画刷的位置在坐标(0,0)处。

2.2.2 创建显示场境

当创建一个显示场境时，用户通常要调用 GDI 的 GetDC()函数并将图形输出窗口的句柄 * 传送给 GDI。在 Windows 内部有一张表，供 Windows 来识别它所维护的各种场境。一个句柄其实就是这张表中的一个数值。用户可以用 Visual Basic 的 hWnd 属性来获取用来绘图的格式(窗口)的句柄，GetDC()返回一个句柄给用户创建好的显示场境，用户再将此句柄传送给所调用的 GDI 图形函数。

2.2.3 释放一个显示场境

因为 Windows 只能保留一定数目的显示场境，所以当用户用结束使用某显示场境后通常应该将它释放，否则，另外一个应用程序可能无法创建显示场境从而不能向屏幕输出。

用户通过调用 GDI 中的 ReleaseDC()函数来释放显示场境，运行良好的应用程序通常在产生图形输出的函数的开头创建一个显示场境，而在函数结尾时便释放此显示场境。一些动画程序当它们启动时创建显示场境而结束时则释放，这比在动画序列的每帧中进行显示场境的创建和释放快得多。

2.2.4 保存和恢复显示场境

用户可以调用 GDI 的 SaveDC()函数将显示场境保存起来。如果用户程序想改变缺省属性的话，这一点很方便。SaveDC()函数将显示场境信息保存在场境堆栈中，它向所存贮的信息返回一个句柄。当要恢复先前保存的显示场境时，调用 GDI 的 RestoreDC()函数即可。

SaveDC()和 RestoreDC()常用于控制多个显示窗的图形应用程序中，通过 SaveDC()与 RestoreDC()来保存和恢复 DC 比每次在使用 DC 时重新设置缺省属性快得多。

2.2.5 兼容的设备场境

兼容的设备场境指是一种仿真的显示场境，它指的是设置成一块位图的内存块。位图就是一个位的矩形阵列，在上面用户可以画图或进行别的图形操作，在动画序列中存贮帧时它特别有用。

为了使位图中的内容能拷贝到屏幕上的应用程序窗口中，位图也应具有许多窗口所具有的 DC 属性，兼容的设备场境指内存中的设备(位图)与显示场境(屏幕上的窗口)相兼容。

2.3 绘图工具

GDI 能比 Visual Basic 提供更广泛更详细的绘图工具。事实上，许多 Windows 编程者认为使用 Visual Basic 的唯一目的就是把它当成利用强大的 GDI 的进入点。

gdi.exe 提供的绘图工具包括画笔、画刷和字体，画笔用来画线和外形，画刷用来给某区域填充彩色，字体提供文本、标题。

当用户要用绘图工具时，需先进行创建和选择，用完后，又必须释放它。绘图工具函数就是 GDI 中专门用来创建、选择和释放绘图工具的例程。绘图属性函数就是专门说明绘图工